On the Efficiency of Backtracking Algorithms for Binary Constraint Satisfaction Problems*

Achref El Mouelhi and **Philippe Jégou** and **Cyril Terrioux**
LSIS - UMR CNRS 6168
Aix-Marseille Université
Avenue Escadrille Normandie-Niemen
13397 Marseille Cedex 20 (France)
{achref.elmouelhi, philippe.jegou, cyril.terrioux}@lsis.org

Bruno Zanuttini
GREYC, Université de Caen Basse-Normandie, CNRS UMR 6072, ENSICAEN
Campus II, Boulevard du Maréchal Juin
14032 Caen Cedex (France)
bruno.zanuttini@unicaen.fr

Abstract

The question of tractable classes of constraint satisfaction problems (CSPs) has been studied for a long time, and is now a very active research domain. However, studies of tractable classes are typically very theoretical. They usually introduce classes of instances together with polynomial time algorithms for recognizing and solving them, and the algorithms can be used only for the new class.

In this paper, we address the issue of tractable classes of CSPs from a different perspective. We investigate the complexity of classical, generic algorithms for solving CSPs (such as Forward Checking). We introduce a new parameter for measuring their complexity and derive new complexity bounds. By relating the complexity of CSP algorithms to graph-theoretic parameters, our analysis allows us to point at new tractable classes, which can be solved directly by the usual CSP algorithms in polynomial time, and without the need to recognize the classes in advance.

Introduction

Constraint Satisfaction Problems (CSPs, (Rossi, van Beek, and Walsh 2006)) constitute an important formalism of Artificial Intelligence (AI) for expressing and efficiently solving a wide range of practical problems. A constraint network (or CSP, abusing words) consists of a set of variables X, each of which must be assigned a value in its associated (finite) domain D, so that these assignments together satisfy a finite set C of constraints.

In general, deciding whether a given CSP has a solution is an NP-complete problem. Hence classical approaches to this problem (and more generally for computing a solution if there is one) are based on backtracking algorithms, whose worst-case time complexity is at best of the order $O(min(n, e).d^n)$ with n the number of variables, e the number of constraints and d the size of the largest domain. To increase efficiency, such algorithms also rely on filtering techniques during search (among other techniques, such as variable ordering heuristics). With the help of such techniques, despite their theoretical time complexity, algorithms such as Forward Checking (Haralick and Elliot 1980) (denoted FC), RFL (for Real Full Look-ahead (Nadel 1988)) or MAC (for "Maintaining Arc Consistency", (Sabin and Freuder 1994)) turn out to be very efficient in practice on a wide range of practical problems.

In a somewhat orthogonal direction, other works have addressed the effectiveness of solving CSPs by defining *tractable classes*. A tractable class is a class of CSPs which can be recognized, and then solved, using polynomial time algorithms. Different kinds of tractable classes have been introduced. Some of them are based on the *structure* of the constraint network, for instance tree-structured networks (Freuder 1982) or more generally, networks of bounded width (Gottlob, Leone, and Scarcello 2000)). Others are based on restrictions of the constraint language. For example, Zero-One-All (ZOA) constraints restrict the compatibility relations to have a certain form (Cooper, Cohen, and Jeavons 1994). More recently, *hybrid classes* (so called because they do not belong to the previous ones) have been proposed, such as the class of instances having the Broken Triangle Property (BTP, (Cooper, Jeavons, and Salamon 2010)).

Unfortunately, most tractable classes rarely occur in practice, which diminishes their interest. In contrast, as evoked above algorithms such as FC, RFL or MAC, whose theoretical complexity is exponential, are the basis of practical systems for constraint solving, and their concrete results are often impressive in terms of computational time.

In this paper, we take a step towards bridging this gap, and address the question of explaining the observed efficiency of algorithms such as FC or RFL. We do so by reevaluating their time complexity using a new parameter, namely the number of maximal cliques in the *micro-structure* of the

*This work was supported by the French National Research Agency under grant TUPLES (ANR-2010-BLAN-0210).

instance (Jégou 1993). Writing $\omega_{\#}(\mu(P))$ for the number of maximal cliques in the micro-structure of CSP a P, we show that the complexity of an algorithm such as FC is in $O(n^2 d \cdot \omega_{\#}(\mu(P)))$. This provides a new perspective on the study of the efficiency of backtracking-like algorithms, by linking it to a well-known graph-theoretic parameter. In particular, reusing known results from graph theory, we revisit some tractable classes of CSPs. The salient feature of these classes is that they are solved in polynomial time by *general-purpose*, *widely used* algorithms, *without the need for the algorithms to recognize the class*. In this respect, our study is very close in spirit to the study by Rauzy about satisfiability problems and the behaviour of DPLL on known tractable instances (Rauzy 1995). Also related to our work is the study by (Chen and Dalmau 2004), who characterize the class of instances which are solved in polynomial-time by a variant of arc-consistency; nevertheless, contrary to our case, the variant in question is not a generic algorithm, since it is incomplete for the class of all CSPs.

The paper is organized as follows. We first introduce notation and recall the definitions and basic properties of the micro-structure. Then we present our complexity analysis of algorithms such as BT and FC. We then point at new tractable classes issued from graph theory, which can be exploited in the field of CSPs. Finally, we give a discussion and perspective for future work.

Preliminaries

We start with some definitions and notation for CSP, then we briefly review a graphical representation of binary CSPs, called *micro-structure*.

Constraint Satisfaction Problems are built from *variables* $x_1, x_2 \ldots$ each associated with a *finite domain* $D(x_i)$.

Definition 1 (constraint) *A* binary constraint c *is a couple* $(S(c), R(c))$*, where the* scope $S(c)$ *of* c *is a couple of variables* (x_i, x_j) *and the* relation $R(c)$ *of* c *is a subset of* $D(x_i) \times D(x_j)$.

Intuitively, the relation of c lists the allowed couples of values for the variables in the scope of c.

Definition 2 (CSP) *A finite binary constraint satisfaction problem (CSP) is a triple* (X, D, C) *where* $X = \{x_1, \ldots, x_n\}$ *is a set of variables,* $D = (D(x_1), \ldots, D(x_n))$ *is a list of finite domains of values (*$D(x_i)$ *is the one of* $x_i \in X$*), and* C *is a set of binary constraints over the variables and domains in* X, D.

We assume that there is at most one constraint per pair of variables $\{x_i, x_j\}$, and we write c_{ij} $(i < j)$ for this constraint. This is without loss of generality since two constraints over the same pair of variables can be merged into one by taking the intersection of their relations.

Definition 3 (assignment, solution) *Given a CSP* (X, D, C)*, an* assignment *of values to* $Y \subseteq X$ *is a set of pairs* $\{(x_i, v_i) \mid x_i \in Y\}$ *with* $v_i \in D(x_i)$ *for all* i*. An assignment to* $Y \subseteq X$ *is said to be* consistent *if all constraints* $c_{ij} \in C$ *with scope* $S(c) = (x_i, x_j) \subseteq Y$ *are satisfied, i.e.,* (v_i, v_j) *is in the relation* $R(c_{ij})$*. Such a consistent assignment is also called a* partial solution *of the CSP. A* solution *is a consistent assignment to* X.

When no confusion can arise, we denote an assignment $\{(x_1, v_1), \ldots, (x_k, v_k)\}$ by the tuple $(v_1, \ldots, v_k)$. We also write n for the number of variables in a CSP, d for the cardinality of the largest domain, and e for the number of constraints.

Given a CSP, the basic question is to decide whether it admits a solution. This problem is well known to be NP-complete. In order to study CSPs and try to circumvent this difficulty, various points of view can be adopted. One of them is the *micro-structure* of an instance, that is, its compatibility graph as we define now. Intuitively, vertices code for values of the CSP, while edges represent the compatibility of pairs of values.

Definition 4 (micro-structure) *Given a CSP* $P = (X, D, C)$*, the* micro-structure *of* P *is the undirected graph* $\mu(P) = (V, E)$ *with:*

- $V = \{(x_i, v_i) : x_i \in X, v_i \in D(x_i)\}$,
- $E = \{ \{(x_i, v_i), (x_j, v_j)\} \mid i \neq j, c_{ij} \notin C$ *or* $c_{ij} \in C, (v_i, v_j) \in R(c_{ij})\}$

In words, the micro-structure of a CSP P contains an edge for all pairs of vertices, except for vertices coming from the same domain and for vertices corresponding to pairs which are forbidden by some constraint. It can easily be seen that the micro-structure of a CSP is an n-partite graph, since there is no edge connecting vertices issued from the same domain.

In this paper, we will study the complexity of CSP algorithms through cliques in the micro-structure. We now briefly review cliques as related to CSPs.

Definition 5 (clique) *A* complete graph *is a simple graph in which every pair of distinct vertices is connected by an edge. A* k-clique *in an undirected graph is a subset of* k *vertices inducing a complete subgraph (all the vertices are pairwise adjacent). A* maximal clique *is a clique which is not a proper subset of another clique.*

Throughout the paper, we write $\omega_{\#}(G)$ for the number of maximal cliques in a graph G, and hence, $\omega_{\#}(\mu(P))$ for the number of maximal cliques in the micro-structure of a CSP P.

In a micro-structure, the vertices of a clique correspond to compatible values which are by construction issued from different domains. Hence, a clique $\{(x_1, v_1), (x_2, v_2), \ldots (x_i, v_i)\}$ in a micro-structure $\mu(P)$ corresponds to a consistent assignment $((x_1, v_1), (x_2, v_2), \ldots (x_i, v_i)) = (v_1, v_2, \ldots v_i)$ for P. This correspondence can be extended to assignments of all the variables.

Proposition 1 *Given a CSP* $P = (X, D, C)$ *and its micro-structure* $\mu(P)$*, an assignment* $(v_1, ..., v_n)$ *to* X *is a solution of* P *if and only if* $\{(x_1, v_1), ..., (x_n, v_n)\}$ *is an* n*-clique of* $\mu(P)$.

It can be seen that the transformation of a CSP P to its micro-structure $\mu(P)$ can be realized in polynomial time. A

polynomial reduction directly follows, from the problem of deciding whether a given CSP (over n variables) has a solution, to the problem of deciding whether a given undirected graph has a clique containing a given number (n) of vertices. While obviously this "clique problem" is also NP-complete, this transformation offers some interests. In particular, it offers a new viewpoint for the study of CSPs, by providing a link to graph-theoretic notions.

This viewpoint has first been exploited by (Jégou 1993), who proposed a new technique for decomposing CSPs. Precisely, (Jégou 1993) introduced a new kind of tractable classes, which is based on known tractable classes for the clique problem (*chordal graphs* (Golumbic 1980)). Later, a similar study has been performed, introducing the notion of *hybrid* tractable classes (Cohen 2003). This work has also been extended by (Salamon and Jeavons 2008), who proposed a new hybrid tractable class which generalizes chordal graphs based on *perfect graphs* (Golumbic 1980). Using this kind of approaches, new tractable classes of CSPs have been proposed, which are defined by *forbidden structures* in the micro-structure (Cooper, Jeavons, and Salamon 2008; 2010).

Nevertheless, as it is generally the case for tractable classes in NP-complete problems, the definition of new tractable classes relies on the design of specific algorithms for recognizing and solving them. What we consider here is another approach, which starts from the algorithms classically used for solving CSPs.

Time Complexity of Backtracking Algorithms

We first briefly review the algorithms of interest here, namely, backtracking (BT), forward checking (FC), and Real Full Look-ahead (RFL), and recall their "classical" complexity analysis. Then we reanalyze their complexity in terms of $\omega_{\#}(\mu(P))$.

Backtracking

The basic algorithm for deciding whether a given CSP has a solution (and computing one in the affirmative) is called *Backtracking* (BT) or *Chronological Backtracking*. It is given by a recursive enumeration procedure, which starts with an empty assignment and in the general case, tries to extend a partial solution $(v_1, v_2, \ldots, v_i)$ by one more variable, to a partial solution $(v_1, v_2, \ldots, v_i, v_{i+1})$.

More precisely, at every stage the algorithm chooses a new variable x_{i+1} and tries to assign values of $D(x_{i+1})$ to x_{i+1}. The only check performed while doing so is that the resulting assignment $(v_1, v_2, \ldots, v_i, v_{i+1})$ is consistent. In the affirmative, BT continues with this new partial solution to a new unassigned variable (called a *future variable*). Otherwise (if $(v_1, v_2, \ldots, v_i, v_{i+1})$ is not consistent), BT tries another value from $D(x_{i+1})$. If there is no such unexplored value, BT is in a dead-end, and then it uninstantiates x_i (it performs a *backtrack*).

It is easily seen that the search performed by BT corresponds to a depth-first traversal of a semantic tree called the *search tree*, whose root is an empty tuple, while the nodes at the i^{th} level are i-tuples which represent the assignments of the variables along the corresponding path in the tree. Nodes in this tree which correspond to partial solutions are called *consistent nodes*, while nodes which correspond to exploring a value v_{i+1} but finding out that $(v_1, v_2, \ldots, v_i, v_{i+1})$ is not consistent, are called *inconsistent nodes*.

Algorithm 1: BT($P = (X, D, C)$: CSP; $t = (v_1, v_2, \ldots v_i)$: tuple)

```
1 if t = (v_1, v_2, ... v_i) is a partial solution then
2     if (i = n) then
3         Exit /* t = (v_1, v_2, ... v_i) is a solution*/
4     else
5         Choose a variable x_{i+1} to assign
6         for v_{i+1} ∈ D(x_{i+1}) do
7             t' ← (v_1, v_2, ... v_i, v_{i+1})
8             BT(P,t')
```

The pseudo-code for BT is shown on Figure 1. As for its (worst-case) time complexity, checking whether $t' = (v_1, v_2, \ldots v_i, v_{i+1})$ is a partial solution consists in checking that the value v_{i+1} assigned to x_{i+1} satisfies the constraints $c_{j(i+1)} \in C$ for all $1 \leq j \leq i$, that is, checking if $(v_j, v_{i+1}) \in R(c_{ji+1})$. So the corresponding number of constraint checks is at most i, which is bounded by n. Moreover, the number of nodes in the search tree is at most $\Sigma_{0 \leq i \leq n} d^i = \frac{d^{n+1}-1}{d-1}$, hence it is in $O(d^n)$. So, the complexity of BT can be bounded by the number of nodes multiplied by the cost at each node, that is, $O(nd^n)$ (we assume $n \leq e$ for simplicity).

Forward Checking

BT can be considered as a generic algorithm, which is not really efficient in practice. In contrast, efficient algorithms generally used in practice are based on processes which achieve some level of filtering at each node, that is, which prune values in the domains of future variables. Note that such algorithms typically also use mechanisms for constraint learning and for non-chronological backtracking, but we do not consider them here.

Filtering is usually based on arc-consistency. One of the most popular approaches is called *Forward Checking*, which is based on arc-consistency without propagation. While after an assignment, BT checks consistency of the current tuple, FC filters the domains of future variables as soon as it assigns a value to the current variable. In that manner, inconsistencies are detected earlier, in fact as soon as such filtering leaves the domain of some future variable empty.

More precisely, assume that a value v_{i+1} has just been assigned to x_{i+1}. Then for all future variables x_j for which there is a constraint $c_{j(i+1)} \in C$, all values in $D(x_j)$ with $(v_j, v_{i+1}) \notin R(c_{j(i+1)})$ (i.e., not consistent with the current partial solution) are removed from $D(x_j)$. Hence from this point on, all values in $D(x_j)$ are consistent with the current partial solution.

The pseudo-code of FC is shown on Figure 2. In essence, the main difference between BT and algorithms which use

Algorithm 2: FC($P = (X, D, C)$: CSP; $t = (v_1, v_2, \ldots v_i)$: tuple)

```
1  if (i = n − 1) then
2      Exit /* t = (v_1, v_2, ... v_i) can be extended to a solution*/
3  else
4      Choose a variable x_{i+1} to assign
5      for v_{i+1} ∈ D(x_{i+1}) do
6          t' ← (v_1, v_2, ... v_i, v_{i+1})
7          Consistent ← True
8          for x_j ∈ X such that i + 1 < j ≤ n and c_{ji+1} ∈ C do
9              D'(x_j) ← {v_j ∈ D(x_j) : (v_j, v_{i+1}) ∈ R_C(c_{ji+1})}
10             if (D'(x_j) = ∅) then
11                 Consistent ← False
12                 Exit-from-loop-for
13         if (Consistent) then
14             P' ← (X, D', C)
15             FC(P', t')
```

filtering techniques, such as FC, is that less nodes are explored in the search tree (inconsistencies are detected earlier), but more work is required at each node. In particular, observe that so as to verify that the CSP has a solution (if it indeed has one), it is not necessary to assign the last variable x_n. Indeed, if its domain is not empty after assigning all variables but x_n and filtering, by construction the remaining values are all consistent with the current partial solution $(v_1, v_2, \ldots v_{n-1})$ and hence, there is at least one solution. Hence the number of nodes in the search tree associated to FC is at most $\Sigma_{0 \leq i \leq n-1} d^i = \frac{d^n - 1}{d-1}$, hence it is in $O(d^{n-1})$. As for the time complexity at each node, each value of the domain of each future variable needs to be checked. The time for doing so is in $O(nd)$ since the current variable x_{i+1} has at most $n-1$ neighboring future variables, for each of which at most d values must be checked. So, the complexity of FC can be bounded by $O(nd \cdot d^{n-1}) = O(nd^n)$, just like for BT.

Real Full Look-Ahead

While from a practical viewpoint, applying other kinds of domain filtering can be relevant, as for example full arc-consistency (algorithm Real Full Look-ahead (Nadel 1988)), such techniques do not yield better worst-case time complexity bounds. For instance, RFL is the same algorithm as FC, except for domain filtering. In RFL, the **for** loop of FC (Line 8), which modifies the domain $D(x_j)$ of each future variable, is replaced by an arc-consistency filtering on the CSP induced by the current assignment and the set of future variables. The time cost for each node is then bounded by the complexity time of AC algorithms (see chapter 3 in (Rossi, van Beek, and Walsh 2006)), that is O(ed^2). Thus, the complexity of RFL is $O(ed^2 d^{n-1}) = O(ed^{n+1})$.

A Note on Dynamic Variable-Ordering Heuristics

We wish to note here that as presented above, BT, FC, and RFL may use a *dynamic* variable ordering, that is, which variable (x_{i+1}) to explore next can typically be decided on each recursive call. However, we do not consider the possibility that after value v_{i+1} is found out to be a dead end, the algorithms may change x_{i+1} to another current variable even if there remains values to explore for it. Indeed, in this paper we stick to the case where the current variable can be changed only when all its values have been explored.

Analysis Based on the Micro-Structure

We now come to the heart of our contribution, namely, a complexity analysis of classical algorithms in terms of parameters related to the micro-structure. In the following, we say that a node of the search tree is a *maximally deep consistent node* if it has no consistent child node. Clearly, a maximally deep consistent node corresponds to a solution or to a consistent assignment which cannot be consistently extended on the next variable.

Proposition 2 *Given a CSP $P = (X, D, C)$, there is an injective mapping from the maximally deep consistent nodes explored by BT in the search tree onto the maximal cliques in $\mu(P)$.*

Proof: Let $(v_1, v_2, \ldots, v_i)$ be a maximally deep consistent node explored by BT. By definition of BT, $(v_1, v_2, \ldots, v_i)$ is a consistent partial assignment, hence for all $1 \leq j, k \leq i$, either there is no constraint with scope (x_j, x_k) in C, or (v_j, v_k) is allowed by the relation $R(c_{ij})$. In both cases there is an edge in $\mu(P)$ by definition of $\mu(P)$, hence $\{(x_1, v_1), \ldots, (x_i, v_i)\}$ forms a clique in $\mu(P)$ and hence, is included in some maximal clique of $\mu(P)$. Write $Cl(v_1, v_2, \ldots, v_i)$ for an arbitrary such maximal clique.

We now show that Cl forms an injective mapping. By construction of BT, if $(v_1, v_2, \ldots, v_i)$ and $(v'_1, v'_2, \ldots, v'_{i'})$ are two maximally deep nodes explored, then they must differ on the value of at least one variable. Precisely, they must differ at least at the point where the corresponding paths split in the search tree, corresponding to some variable x_j assigned to some value on one path, and to some other value on the other one[1]. Since there are no edges in $\mu(P)$ connecting two values of the same variable, there cannot be a maximal clique containing both $(v_1, v_2, \ldots, v_i)$ and $(v'_1, v'_2, \ldots, v'_{i'})$, hence Cl is an injective mapping. □

Using this property, we can easily bound the number of nodes in a search tree induced by a backtracking search, and its time complexity, in terms of the micro-structure. As is common, we assume that a constraint check (deciding $(v_i, v_j) \in R(c_{ij})$) requires constant time.

Proposition 3 *The number of nodes $N_{BT}(P)$ in the search tree developed by BT for solving a given CSP $P = (X, D, C)$, satisfies $N_{BT}(P) \leq nd \cdot \omega_{\#}(\mu(P))$. Its time complexity is in $O(n^2 d \cdot \omega_{\#}(\mu(P)))$.*

Proof: First consider the number of consistent nodes. Because any node in the search tree is at depth at most n and the path from the root to a consistent node contains only consistent nodes, as a direct corollary of Proposition 2 we obtain

[1] We use at this point the assumption that the algorithms explore all the values of a variable before reordering the future variables.

that the search tree contains at most $n \cdot \omega_{\#}(\mu(P))$ consistent nodes. Now by definition of BT, a consistent node can have at most d children (one per candidate value for the next variable), and inconsistent nodes have none. It follows that the search tree contains at most $nd \cdot \omega_{\#}(\mu(P))$ nodes of any kind.

The time complexity follows directly, since each node corresponds to extending the current partial assignment to one more variable (x_{i+1}), which involves at most one constraint check per other variable (check $c_{j(i+1)}$ for each x_j already assigned).□

It can be seen that in the statement of Proposition 3 and in the forthcoming ones, the number of maximal cliques $\omega_{\#}(\mu(P))$ could be replaced by the number of maximal cliques *of size at most* $n-1$. This is because as soon as a path is explored which is contained in an n-clique, that is, in a solution, no backtracking will occur further that this path. Importantly, this means that our study may show a class of CSPs to be solved in polynomial time by BT even if some CSPs have an exponential number of solutions (hence of maximal cliques *of size* n). Anyway, we do not pursue this idea in the rest of this paper.

We now turn to forward checking. Clearly enough, Proposition 2 also holds for FC, and again we can derive the size of the search tree and time complexity of FC in terms of the micro-structure.

Proposition 4 *The number of nodes $N_{FC}(P)$ in the search tree developed by FC for solving a given CSP $P = (X, D, C)$, satisfies $N_{FC}(P) \leq n \cdot \omega_{\#}(\mu(P))$. Its time complexity if in $O(n^2 d \cdot \omega_{\#}(\mu(P)))$.*

Proof: As in the proof of Proposition 3, it follows from Proposition 2 that the search tree contains at most $n \cdot \omega_{\#}(\mu(P))$ consistent nodes. Since by definition of FC, all nodes are consistent, we get that this is also the size of the search tree.

The time complexity follows from the fact that when the current partial solution is extended to one more variable, the domains of at most n future variables must be filtered, each one in time $O(d)$ since checking one value amounts to a constraint check with the newly assigned value. □

Finally, since FC and RFL differ only in the filtering process at each node, which is an arc consistency process ($O(ed^2)$) at each node for RFL, we get the following.

Proposition 5 *The time complexity of RFL for solving a CSP $P = (X, D, C)$ is in $O(ned^2 \cdot \omega_{\#}(\mu(P)))$.*

It is important to observe that the relative size of the search trees of BT and FC are the same with the classical analysis and with the expressions which we have derived. Namely, in the worst case, the search tree of BT is d times larger than that of FC on the same instance. Because this is true despite the fact that the bounds are derived in a different manner, this suggests that our bounds are tight (as worst-case bounds).

A Few Tractable Classes for Backtracking

The number of cliques in a graph can grow exponentially with the size of the graph (Wood 2007), and so can the number $\omega_{\#}(G)$ of *maximal* cliques in a graph G (Moon and Moser 1965). However, for some classes of graphs, the number of maximal cliques can be bounded by a polynomial in the size of the graph. If the micro-structure $\mu(P)$ of a (family of) CSP P belongs to one of these classes, then our analysis in the previous section allows to conclude that P can be solved in polynomial time by classical backtracking algorithms. In this section, we study several such classes of graphs in terms of their relevance to constraint satisfaction problems.

Triangle-Free, Bipartite, and Planar Graphs

Recall that a *k-cycle* in a graph $G = (V, E)$ is a sequence $(v_1, v_2, \ldots v_{k+1})$ of vertices satisfying $\forall i, 1 \leq i \leq k, \{v_i, v_{i+1}\} \in E$, $\forall i, j, 1 \leq i < j \leq k, v_i \neq v_j$, and $v_1 = v_{k+1}$.

Definition 6 (triangle-free) *A* triangle-free *graph is an undirected graph with no* 3*-cycle.*

It is easily seen that the number of maximal cliques in a triangle-free graph is exactly its number of edges E. Indeed, each edge is a clique, and no greater clique exists by definition. Hence by our analysis, if a class of CSPs has a triangle-free micro-structure, algorithms BT, FC, and RFL correctly solve them in polynomial time. Note however that this is quite a degenerate case, since except for instances over at most two variables, instances with a triangle-free micro-structure are inconsistent.

Another degenerate but illustrative case is the following.

Definition 7 (bipartite) *A graph is* bipartite *if and only if it does not contain an odd cycle.*

Again, a bipartite graph cannot contain any clique of more than two variables, and hence no partial assignment to more than three variables will ever be considered by BT (hence it obviously runs in time $O(d^3)$).

We now turn to a more interesting class, which also essentially contains inconsistent CSPs, but for which our analysis gives a better time complexity than the classical one.

Definition 8 (planar) *A* planar *graph is a graph which can be drawn in such a way that no two edges cross each other.*

(Wood 2007) proved that the number of cliques in a planar graph is at most $8(|V| - 2)$. Since the micro-structure $\mu(P)$ of a CSP P contains at most nd vertices, it follows that if $\mu(P)$ is planar, then its number of maximal cliques $\omega_{\#}(\mu(P))$ is at most $8(nd - 2)$. Using our results in the previous section (Propositions 3–5), we immediately get the following.

Theorem 1 *Let Pl denote the class of all CSPs whose micro-structure is planar. Then instances in Pl are solved in time*

- $O(n^2 d \cdot \omega_{\#}(\mu(P))) = O(n^3 d^2)$ *by BT or FC,*
- $O(ned^2 \cdot \omega_{\#}(\mu(P))) = O(n^2 ed^3)$ *by RFL.*

Recall that a planar graph cannot contain as a minor, a 5-clique or the graph $K_{3,3}$ (i.e. a complete bipartite graph with three vertices connected to three other vertices). It follows in particular that any CSP in Pl over at least five variables is inconsistent. Hence again this class is a little degenerate, however a classical analysis states that, e.g., BT solves these instances in time $O(d^5)$. In case d is large, this is looser that $O(n^3d^2)$.

CSG Graphs

We finally turn to the class of *CSG graphs*, which has been introduced by (Chmeiss and Jégou 1997) and which generalizes the class of chordal graphs. Given a graph (V, E) and an ordering $v_1, \ldots, v_{|V|}$ of its vertices, we write $N^+(v_i)$ for the *forward neighborhood* of v_i, that is, $N^+(v_i) = \{v_j \in V | \{v_i, v_j\} \in E, i < j\}$. For $V' \subseteq V$, we write $G(V')$ for the graph *induced* by E on V', namely, $G(V') = (V', E')$ where $E' = \{\{x, y\} \mid x, y \in V' \text{ and } \{x, y\} \in E\}$.

Definition 9 (CSG graphs) *The class of graphs CSG^k is defined recursively as follows.*

- *CSG^0 is the class of complete graphs.*
- *Given $k > 0$, CSG^k is the class of graphs $G = (V, E)$ such that there exists an ordering $\sigma = (v_1, ..., v_{|V|})$ of V satisfying that for $i = 1, \ldots, |V|$, the graph $G(N^+(v_i))$ is a CSG^{k-1} graph.*

The class of CSG graphs generalizes the class of complete graphs (CSG^0 graphs) and the class of chordal graphs (CSG^1 graphs). Like chordal graphs, CSG graphs have nice properties. For instance, they can be recognized in polynomial time. Moreover, Chmeiss and Jégou have proved that CSG^k graphs have at most $|V|^k$ maximal cliques, and they have proposed an algorithm running in time $O(|V|^{2(k-1)}(|V| + |E|))$ for finding all of them.

These two algorithms qualify the class of all CSPs which have a CSG^k micro-structure as a tractable class for any fixed k. We are however able to show that even a *generic* algorithm such as BT, FC, or RFL runs in polynomial time on such CSPs, without even the need to recognize membership in this class, nor to compute the micro-structure. Again, the result follows from the number of maximal cliques together with Propositions 3–5.

Theorem 2 *Given any integer k, the class of all CSPs which have a CSG^k micro-structure is solved in time*

- *$O(n^2d \cdot \omega_{\#}(\mu(P))) = O(n^{k+2}d^{k+1})$ by BT and FC,*
- *$O(ned^2 \cdot \omega_{\#}(\mu(P))) = O(n^{k+1}ed^{k+2})$ by RFL.*

Observe that even the time complexities are better than those of the dedicated algorithm. The latter computes the micro-structure and enumerates all maximal cliques until exhaustion, or an n-clique is found. So it has a time complexity in $O((nd)^{2(k-1)}(nd + n^2d^2)) = O((nd)^{2k})$. Nevertheless, we can note that this algorithm is defined for general CSG^k graphs while micro-structures of CSP are very particular graphs.

While the classes of planar and of CSG graphs are not comparable, CSG graphs are generally speaking less restrictive than planar graphs. For instance, it is possible to have CSG graphs with n-cliques for any value of n, contrary to the case of planar graphs. In particular, there are consistent CSPs with a CSG^k micro-structure. It is the case for CSG^0 which are consistent CSPs with monovalent domains (one value per domain). Moreover, CSPs which have a CSG^1 micro-structure can be consistent or not and it is easy to build a CSP with several solutions, which corresponds to a collection of cliques of size n. However, this set of classes of CSPs still has to be studied in detail for assessing its practical interest.

Discussion and Perspectives

We have investigated the time complexity of classical, generic algorithms for solving CSPs under a new perspective. Our analysis expresses the complexity in terms of the number of cliques in the micro-structure of the CSP to be solved. Our analysis reveals that essentially, backtracking and forward checking visit each maximal clique in the micro-structure at most once.

From this analysis we derived tractable classes of CSPs, which can be solved by classical algorithms in polynomial time, *without the need to recognize that the instance at hand is in the class*. Though the results obtained so far in this manner are of limited practical interest, they shed a new light on the analysis of CSPs.

The first perspective of this work is to investigate more classes of graphs with polynomially many maximal cliques. Of particular interest here is the study by (Rosgen and Stewart 2007), who precisely characterize these classes of graphs in terms of intersection graphs.

Another important perspective is to relate our analysis to tractable classes obtained in different manners. It should be rather clear that our classes are orthogonal to classes based on structure. For instance, there is no reason why a tree-structured CSP would not have an exponential number of cliques. More generally, classes based on structure typically require dedicated algorithms, and generic backtracking algorithms are polynomial on them only if they proceed, e.g., along a specific variable ordering. As concerns classes defined by a constraint language, it may be the case that some can be captured by our analysis, just as is the case for the satisfiability problem (Rauzy 1995). Finally, hybrid classes are much closer in spirit, and it is a short-term perspective to investigate in depth the link between such classes and our analysis.

Finally, an important perspective is to extend our study to n-ary constraint satisfaction problems. For this we plan in particular to investigate the standard reductions from the n-ary to the binary case.

Acknowledgments

Philippe Jégou would like to thank Maria Chudnowsky for their fruitful discussion, about graphs theory, perfect graphs and links with classes of graphs related to the clique problem, and the links with the microstructure of CSPs.

References

Chen, H., and Dalmau, V. 2004. (smart) look-ahead arc consistency and the pursuit of csp tractability. In Wallace, M., ed., *CP*, volume 3258 of *Lecture Notes in Computer Science*, 182–196. Springer.

Chmeiss, A., and Jégou, P. 1997. A generalization of chordal graphs and the maximum clique problem. *Information Processing Letters* 62:111–120.

Cohen, D. A. 2003. A New Classs of Binary CSPs for which Arc-Constistency Is a Decision Procedure. In *Proceedings of CP 2003*, 807–811.

Cooper, M.; Cohen, D.; and Jeavons, P. 1994. Characterising Tractable Constraints. *Artificial Intelligence* 65(2):347–361.

Cooper, M.; Jeavons, P.; and Salamon, A. 2008. Hybrid tractable CSPs which generalize tree structure. In *Proceedings of ECAI 2008*, 530–534.

Cooper, M.; Jeavons, P.; and Salamon, A. 2010. Generalizing constraint satisfaction on trees: hybrid tractability and variable elimination. *Artificial Intelligence* 174:570–584.

Freuder, E. 1982. A Sufficient Condition for Backtrack-Free Search. *JACM* 29 (1):24–32.

Golumbic, M. 1980. *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, New York.

Gottlob, G.; Leone, N.; and Scarcello, F. 2000. A Comparison of Structural CSP Decomposition Methods. *Artificial Intelligence* 124:343–282.

Haralick, R., and Elliot, G. 1980. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence* 14:263–313.

Jégou, P. 1993. Decomposition of Domains Based on the Micro-Structure of Finite Constraint Satisfaction Problems. In *Proceedings of AAAI 93*, 731–736.

Moon, J. W., and Moser, L. 1965. On cliques in graphs. *Israel Journal of Mathematics* 3:23–28.

Nadel, B. 1988. *Tree Search and Arc Consistency in Constraint-Satisfaction Algorithms*. In *Search in Artificial Intelligence*. Springer-Verlag. 287–342.

Rauzy, A. 1995. Polynomial restrictions of SAT: What can be done with an efficient implementation of the Davis and Putnam's procedure. In Montanari, U., and Rossi, F., eds., *Proc. International Conference on Principles of Constraint Programming (CP 1995)*, 515–532. Springer Verlag.

Rosgen, B., and Stewart, L. 2007. Complexity results on graphs with few cliques. *Discrete Mathematics and Theoretical Computer Science* 9:127–136.

Rossi, F.; van Beek, P.; and Walsh, T. 2006. *Handbook of Constraint Programming*. Elsevier.

Sabin, D., and Freuder, E. 1994. Contradicting Conventional Wisdom in Constraint Satisfaction. In *Proc. of ECAI*, 125–129.

Salamon, A., and Jeavons, P. 2008. Perfect Constraints Are Tractable. In *Proceedings of CP*, 524–528.

Wood, D. R. 2007. On the maximum number of cliques in a graph. *Graphs and Combinatorics* 23:337–352.